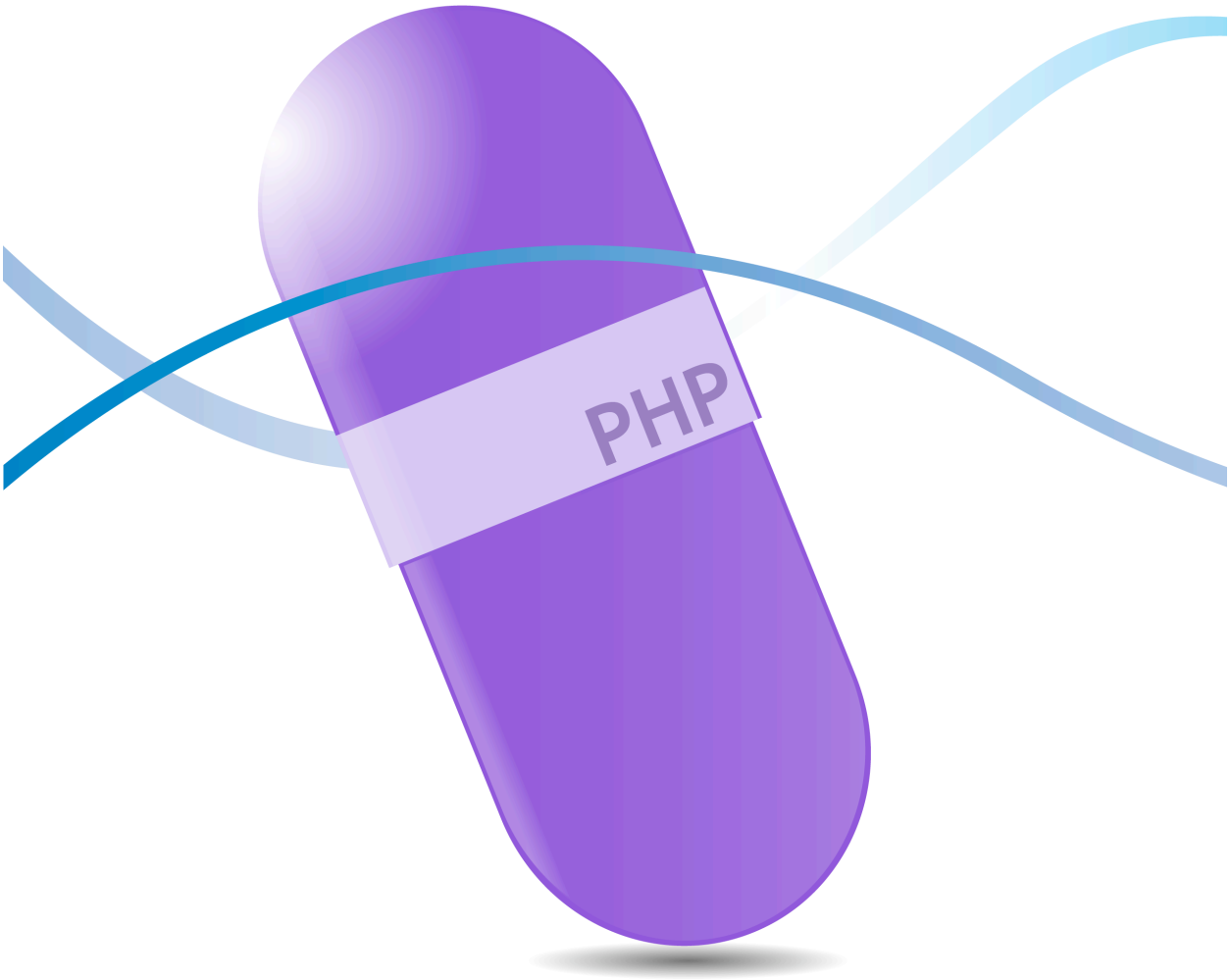
SmartPill

PHP Edition™



scodigo

User Manual

Version 1.5

Published January 13, 2008

Copyright © 2008 Scodigo, Inc.
Visit http://www.scodigo.com for more info.

This product includes PHP software, freely available at www.php.net/software.

Thanks to Tim Dietrich from FileMaker Addict for help with the user manual.
filemakeraddict.blogspot.com

Thanks to Pumbaa80 for the smiley.
commons.wikimedia.org/wiki/Image:Smiley.svg

Contents

Support....4
System Requirements....4
Trial License Available....4
What Is SmartPill?....5
What's New In Version 1.5?....6
Installation....7
  Macintosh Installation....7
  Windows Installation....7
  Server Installation....8
  Web Publishing Installation....8
  PHP Support Folder....9
  License.txt File....9
  php.ini File....9
  Using PHP Extensions....10
  Using FileMaker's AutoUpdate Plug-in....11
Parameters....13
Functions....14
  PHP_Register....14
  PHP_Version....15
  PHP_Control....16
  PHP_Timeout....17
  PHP_GetLastError....18
  PHP_Execute....19
  PHP_Execute_To_Container....21
PHP Include Path....22
PHP FileMaker Extension....23
  Evaluate Function....23
  Script Functions....25
  Error Function....27
  SQL Functions....29
  Variable Functions....31

Support

We're here to help! Please contact us if you have any issues with installation or questions regarding how the plug-in works. We're also very interested in feedback and would like to hear about how you're using SmartPill.

The best way to contact us for support is by using our plug-in support request form located at http://www.scodigo.com/support. You can also send email to scodigo.support@scodigo.com.

In addition, we have forums dedicated specifically to installation and support: http://www.scodigo.com/forums.

System Requirements

FileMaker 8 or higher (including version 9).

Macintosh PowerPC or Intel
OSX version 10.3.9 for PowerPC and 10.4.5 with Intel

Pentium III 500 MHz or higher
Windows 2000 (Service Pack 4)
Windows XP (Service Pack 2)

Although formal testing has not been performed, no issues have been encountered when using Microsoft Terminal Services or Citrix.

Trial License Available

Our plug-ins will function without restriction for 30 minutes each time FileMaker is launched. After this grace period is exceeded, registration is required. Please visit http://www.scodigo.com/support/trial-license if you would like a fully functional 30-day license.

What Is SmartPill?

SmartPill does one simple but very powerful thing, it allows you to run PHP code and return the results back to FileMaker. Putting the power of PHP inside FileMaker opens a whole new world of possibilities. Visit PHP's website to view a complete list of functions: http://www.php.net/manual/en/funcref.php.

Here are just a few of the things you can do:

- Send and receive email
- Send and receive files using FTP
- Get URLs (with SSL support)
- Read and write to Active or Open Directory with LDAP functions
- Communicate with web services
- Encrypt and decrypt your data
- Execute shell commands and return the results
- XML, XSL and XPath functions
- Read and write to files on disk
- Connect to MySQL databases
- Utilize the power of regular expressions
- Access an extensive set of string functions
- Read and write zip files
- Create and manipulate images and return the results to a container field
- Read EXIF data from image files

SmartPill also includes a powerful set of PHP functions that allow you to interact with FileMaker from within your PHP code. Here a some of the things you can do with our PHP FileMaker extension:

- Evaluate FileMaker calculations
- Perform scripts
- Execute SQL
- Create application level variables

What's New In Version 1.5?

- Container fields can now be set directly using the new PHP_Execute_To_Container function. Now you can use PHP functions to retrieve (or even create) images, documents and other files and store them directly in the database. Most file types are supported and preview images are generated for the following graphic files: bmp, gif, giff, jpg, jpeg, pct, pdf, png, psd, tif, tiff.
- The Windows version is now distributed as a single ".fmx" plug-in file. This greatly simplifies installation on the Windows platform and will now work with the FileMaker auto-update function for plug-in installation and update on client machines.
- The MySQL extension is now compiled into the Windows version and the PostgreSQL and Microsoft SQL (MSSQL) extensions are now compiled in for both Mac and Windows versions. The PDO extension is also now included.
- New fm_sql_select and fm_sql_execute functions allow you to perform SQL queries internally without the need for ODBC drivers or any type of configuration. Now you can easily access the power and flexibility of SQL.
- New variable functions allow you to manage variables that are stored globally at the application level which means they can be accessed from any file.
- New fm_script_control function allows script triggering to be turned on and off. This is very helpful for situations when you don't want script triggering to occur.
- New fm_get_last_error function allows you to check for errors returned by other "fm" functions that are included in the PHP extension.
- New fm_set_error function allows you to return user-defined errors from within your PHP code. Errors set with this function can be read with using the PHP_GetLastError function.
- The PHP_GetLastError function now includes an optional selector parameter allowing you to specify the return of just the error number or just the error description.
- The calendar and wddx extensions, which were previously only available in the Windows version, are now compiled in for the Mac versions.
- Now includes PHP version 5.2.4.

Installation

SmartPill comes in 3 versions, so it's important that you install the correct version for your operating system. Use the following table to determine the correct version:

Operating System	File Name
Macintosh OSX PowerPC (PPC)	PHP_PPC.fmplugin
Macintosh OSX Intel	PHP_Intel.fmplugin
Windows	PHP.fmx

Macintosh Installation

Installation for both Macintosh platforms is the same; simply place the appropriate file (PPC or Intel version) in the Extensions folder. The Extensions folder is located inside the FileMaker application folder or in a new alternative location for FileMaker 9. The typical location is here (assuming FileMaker Pro 8.5 Advanced):

```
/Applications/FileMaker Pro 8.5 Advanced/Extensions/
```

New alternative location for FileMaker Pro 9:

```
/Users/UserName/Library/Application Support/FileMaker/
Extensions/
```

Windows Installation

Place the PHP.fmx file in the Extensions folder. The Extensions folder is located inside the FileMaker application folder or in a new alternative location for FileMaker 9. The typical location is here (assuming FileMaker Pro 8.5 Advanced):

```
C:\Program Files\FileMaker\FileMaker Pro 8.5 Advanced\
Extensions\
```

New alternative location for FileMaker Pro 9:

XP: `C:\Document Settings\UserName\Local\Settings\ApplicationData\FileMaker\Extensions\`

Vista™: `C:\Users\UserName\AppData\Local\FileMaker\Extensions\`

Server Installation

When SmartPill is installed on the server, certain calculations may resolve "server-side" but there's limited documentation from FileMaker regarding exactly how this works. With FileMaker Server 9, we can now run scripts using schedules which opens up a world of possibilities, especially with the help of SmartPill.

To install on the Server, put the plug-in file and the PHP Support folder (with License.txt file) here:

Windows: `[drive]:\Program Files\FileMaker\FileMaker Server \Database Server\Extensions\`

MacOS: `/Library/FileMaker Server/Database Server/Extensions/`

Web Publishing Installation

SmartPill can be installed on the Web Publishing Engine and will work with both Instant Web Publishing and Custom Web Publishing. One major caveat for use with web publishing is that plug-ins are only loaded once for all web sessions and it's possible that the plug-in will return error 200 "Plug-in is busy" if another session happens to be using the plug-in at the same time.

Scripting and calculations that rely on the plug-in should be written to handle error 200 "Plug-in is busy" and we recommend that you test your solutions thoroughly with multiple simultaneous users.

To install on the web publishing engine, put the plug-in file and the PHP Support folder (with License.txt file) here:

Windows: `[drive]:\Program Files\FileMaker\FileMaker Server \Web Publishing\publishing-engine\wpc\Plugins\`

MacOS: `/Library/FileMaker Server/Web Publishing/publishing-engine/wpc/Plugins/`

PHP Support Folder

If you would like to use a license file so that the plug-in is automatically registered, or if you would like to use a custom php.ini file, you will need to create a folder named “PHP Support” inside the Extensions folder. We also recommend using this folder as a place to store other PHP files or libraries like PEAR.

The plug-in looks for the PHP Support folder relative to the plug-in file. This means that if the plug-in is installed in the new alternative location for FileMaker 9, the plug-in will look for the “PHP Support” folder inside that Extensions folder, NOT in the Extensions folder located in the application folder.

License.txt File

As an alternative to registering the plug-in via a script step (which is explained in the “Functions” section later in this manual), you can use a license file to automatically register the plug-in when FileMaker starts up. To do so, create a text file named “License.txt” and place it in the “PHP Support” folder. (For information about this folder, see the “PHP Support Folder” section above.)

The contents of the license.txt file should follow this format:

```
Name: Joe Smith
Company: Smith Manufacturing, Inc.
License Key: GKQJN2-MHY2TX-1DD1HI-WP0201-0100F3-3E3201
```

php.ini File

Advanced users may want to use a customized PHP initialization file (named “php.ini”) instead of the ini file that was used when the plug-in was compiled by Scodigo. To do so, place your custom php.ini file in the “PHP Support” folder. (For information about this folder, see the “PHP Support Folder” section above.)

We’ve included a sample php.ini file with the distribution and recommend that you use it as a starting point when creating your customized ini file.

There are two very important settings that must be set in the php.ini file:

```
max_execution_time = 0
max_input_time = 0
```

If you use a php.ini file with settings other than these, FileMaker will crash!

Using PHP Extensions

Important: Loading PHP extensions is not supported in version 1.5.

We are looking into the possibility of releasing a separate version of SmartPill, where PHP is not compiled into the plug-in. This would allow extensions to be loaded.

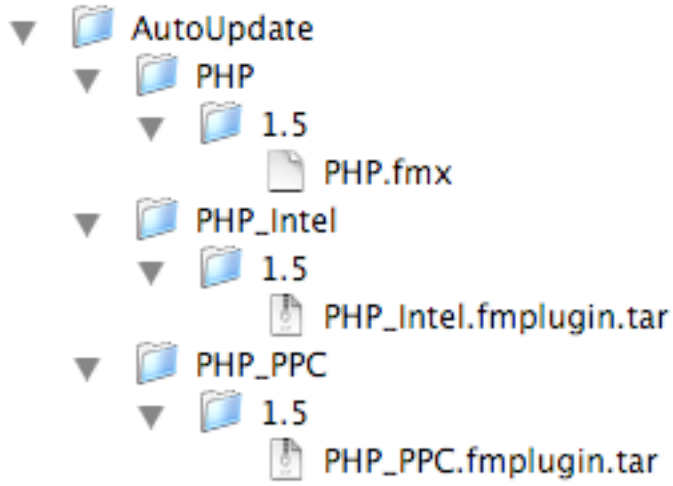
In the meantime, if you need to load extensions, please use SmartPill version 1.0.

We've worked hard to include the most popular extensions and we are open to requests. If loading extensions is important to you, please let us know.

Using FileMaker's AutoUpdate Plug-in

FileMaker includes an AutoUpdate plug-in that allows plug-ins to be installed and updated on client machines. This provides an easy way for administrators to maintain the correct plug-ins for their solutions. For more details, please download FileMaker's documentation here: http://www.filemaker.com/downloads/pdf/fms9_autoupdate_en.pdf.

Here's our recommended folder structure:



You can download .tar versions of the Mac plug-ins here: http://www.scodigo.com/support/downloads.

To calculate the version:

```
Max ( 0 ; GetAsNumber ( LeftValues ( PHP_Version ;
1 ) ) )
```

Returns:
1.5 if the plug-in version is 1.5.0 or 1.51 if the plug-version is 1.5.1

To update the plug-in to version 1.5:

```
Let (
  [
    platform = Case ( Get ( SystemPlatform ) = -1 ;
"_PPC" ; Get ( SystemPlatform ) = 1 ; "_Intel" )
  ] ;
  FMSAUC_UpdatePlugIn( "PHP" & platform & " 1.5" )
)
```

Returns:
0 if there's no error or one of the following errors (from FileMaker's documentation):

-1 The file to be downloaded is missing from the temporary folder.

-2 The Extensions\Saved folder to contain the backup of the outdated plug-in or support file couldn't be created on the client computer .

-3 The file to be replaced on the client computer couldn't be deleted from the Extensions folder.

-4 The file to be replaced couldn't be moved to the Extensions\Saved folder.

-5 The downloaded file can't be copied to the Extensions folder.

-6 The download file must be a plug-in file.

3 The Auto Update plug-in is disabled in the FileMaker Server Admin Console.

5 The download file can't be found in the AutoUpdate folder on the FileMaker Server computer.

6 An error occurred on the computer running FileMaker Server as the file was being downloaded.

100 The external function definition for FMSAUC_UpdatePlugIn contains an invalid or empty parameter.

101 The function call from the client computer to the computer running FileMaker Server failed. The server computer might be running a previous version of FileMaker Server.

Parameters

Many of the SmartPill's functions require you to include parameters. In some cases, all of the parameters are required and in other cases, only some of the parameters are required. In addition, some functions don't require any parameters at all. Here's an explanation of how to tell which parameters are required.

Using the PHP_Register as an example, the syntax for the function will be listed as follows:

```
PHP_Register( {licenseOwnerName ; licenseOwnerCompany ;
licenseKey} )
```

This indicates that the PHP_Register function requires either **all 3 parameters** or **no parameters**. Because all 3 parameters are enclosed in curly braces, they are optional.

As a second example, let's look at the syntax for the PHP_Execute function:

```
PHP_Execute( phpCode {; keepMemory {; timeoutSeconds}} )
```

The PHP_Execute function is a bit trickier; it will accept **1, 2 or 3 parameters**. The first parameter is required because it's not enclosed in curly braces. The second two parameters are optional because they are enclosed in curly braces. The third parameter is also optional because it's enclosed in it's own set of curly braces.

As a final example, let's look at the PHP_Version function:

```
PHP_Version
```

The PHP_Version function **doesn't accept any parameters**. It simply returns the version information so no parameters are required.

Functions

PHP_Register({licenseOwnerName ; licenseOwnerCompany ; licenseKey})

This function allows the plug-in to be registered. Using PHP_Register with no parameters will return the current registration information. The plug-in will function for 30 minutes without being registered each time FileMaker is launched. After a 30 minute grace period, attempting to use any function other than PHP_Version will result in error 100 (plug-in is not registered).

Examples

```
// valid registration
PHP_Register( "Beta Tester" ; "Beta" ; "GKQJN2-
MHY2TX-1DD1HI-WP0201-0100F3-3E3201" )
```

Returns:
0 (no error, plug-in is registered)

```
// get registration info
PHP_Register
```

Returns:
Name: Beta Tester
Company: Beta
License Type: Temporary
User Count: 1
Expiration Date: 01-31-2007

```
// invalid registration
PHP_Register( "Boogie Man" ; "BMI" ;
"123456-123456-123456-123456-123456-123456" )
```

Returns:
101 Invalid registration (error, plug-in is not registered)

PHP_Version

This function returns both the plug-in version and the PHP version.

Examples

```
// check the current setting
PHP_Version
```

Returns:
1.5.0
PHP Version 5.2.4
Copyright © 2007 Scodigo, Inc. Visit http://www.scodigo.com for more info. This product includes PHP software, freely available from http://www.php.net/software.

PHP_Control({enableDisable ; password})

This function allows the plug-in to be disabled. It is useful in situations where access to the plug-in needs to be restricted.

To disable, pass in a 0 as the first parameter and a password of your choosing as the second parameter. Once the plug-in is disabled, it can only be enabled by passing in a 1 as the first parameter and the correct password as the second parameter. Using PHP_Control with no parameters will return the current setting: 1 if enabled, 0 if not enabled.

Examples

```
// disable the plug-in
PHP_Control( 0 ; "secret" )
```

Returns:
0 (no error, plug-in is disabled)

```
// check the current setting
PHP_Control
```

Returns:
0 (plug-in is disabled)

```
// attempt to enable the plug-in with the wrong password
PHP_Control( 0 ; "boogie man" )
```

Returns:
105 Password to enable is wrong (error, plug-in is still disabled)

```
// enable the plug-in with the correct password
PHP_Control( 1 ; "secret" )
```

Returns:
0 (no error, plug-in is enabled)

PHP_Timeout({seconds})

This function allows you to set the timeout duration for PHP execution. For example, if you want to download a large file from an FTP server, you may need to increase the timeout duration.

To set the timeout duration, pass in the number of seconds.

Using PHP_Timeout with no parameters will return the current setting. The default setting is 60 seconds.

Note that you can also specify the timeout duration for individual calls to PHP using the optional third parameter (“timeoutSeconds”) of the PHP_Execute and PHP_Execute_To_Container functions.

Examples

```
// check the current setting
PHP_Timeout
```

Returns:
60 (assuming the default setting hasn’t been changed)

```
// set time out to 10 seconds
PHP_Timeout( 10 )
```

Returns:
0 (no error, timeout setting is changed)

PHP_GetLastError({selector})

This function provides an easy way to check for plug-in errors.

If there is no error, this function will return 0, otherwise it will return an error number with a description of the error.

Pass in a 1 if you only want the error number to be returned. Pass in a 2 to return only the error description.

See the fm_set_error function in the PHP FileMaker Extension section below for information on how to return user-defined errors from your PHP code.

Examples

```
PHP_GetLastError
```

Returns:
0 (assuming no error)

Possible Errors:

0 No error

100 Plug-in is not registered

101 Invalid registration

102 License is for an older version of the plug-in (upgrade required)

103 License is expired (new license required)

104 Plug-in is currently disabled

105 Plug-in could not be enabled (incorrect password)

106 Invalid parameter

200 Plug-in is busy (main)

201 Timeout

PHP_Execute(phpCode {; keepMemory {; timeoutSeconds }})

This is the main function. It allows PHP code to be executed and for the results to be returned to FileMaker.

The default behavior for the plug-in is to clear any previous variables, objects, etc. from memory before starting a new execution. In some cases, you may not want memory to be cleared.

The first parameter is the PHP code that is to be executed.

The second parameter is used to indicate if you would like to maintain the values of the PHP code execution in memory. This allows subsequent calls to PHP to access the values that were set by the previous call. Pass in a 1 as the second parameter to “keepMemory”.

The third parameter can be used to control the timeout duration for a single execution. Simply pass in the number of seconds for the timeout.

Examples

```
// the obligatory hello world
PHP_Execute( "echo 'hello world';" )
```

Returns:
hello world

```
// set $my_var to 'hello world'
PHP_Execute( "$my_var = 'hello world';" ) &

// use the PHP var_dump function to show that $my_var
// still exists when the keepMemory parameter is
// set to 1
PHP_Execute( "var_dump($my_var);" ; 1 ) &

// now execute the same code with keepMemory set to 0
// and notice that $my_var is now NULL
PHP_Execute( "var_dump($my_var);" ; 0 )
```

Returns:
string(11) "hello world" NULL

```
// use the PHP sleep function to show what happens when
// the timeoutSeconds parameter is set to a duration
// that's less than the execution time
PHP_Execute( "sleep( 2 );" ; 0 ; 1 )
```

Returns:
201 Timeout

PHP_Execute_To_Container(phpCode ; fileName {; keepMemory {; timeoutSeconds }})

This function allows you to save the results of your PHP code directly to a FileMaker container field.

Other than the fileName parameter, this function is identical to PHP_Execute.

The file extension should be included in the file name, for example, “scodigo-logo.**png**”. If the plug-in recognizes the file extension and considers it to be a graphic file, it will generate a preview for display. The supported graphic file types are bmp, gif, giff, jpeg, jpg, pct, pdf, png, psd, tif, tiff. Files are stored in the database just as if you had imported them without “Store only a reference to the file” checked.

Examples

```
$url = 'http://www.scodigo.com/images/scodigo-logo.png';

$file = file_get_contents($url);

if (! $file) {
     die("Could not load file.");
}

echo $file;
```

Returns:
scodigo-logo.png file with preview

PHP Include Path

PHP uses a setting named **include_path** to store a list of directories where functions like require() and include() will look for files. When SmartPill starts up, it will append the path to the PHP Support folder and an includes folder in the PHP Support folder, to the include_path. This gives you a reliable place to put your include files along with the convenience of not having to set the include_path with each execution.

For example:

```
/Users/UserName/Library/Application Support/FileMaker/
Extensions/PHP Support:/Users/UserName/Library/Application
Support/FileMaker/Extensions/PHP Support/includes
```

Keep in mind that the PHP Support folder isn't created by default, so if you need it, you will have to create it. (See the **Installation** section earlier in this manual for details on the PHP Support folder.)

PHP FileMaker Extension

SmartPill includes a PHP extension that adds a number of powerful FileMaker-related functions to PHP. These are actual PHP functions that can be used in your PHP code. With these functions you can return the results of any FileMaker calculation, trigger scripts, execute SQL against your FileMaker data, create globally available variables, and define your own errors.

Evaluate Function

string **fm_evaluate** (string $expression)

Description

Evaluates the specified expression using FileMaker's calculation engine and returns the results.

Parameters

expression (required)
Any expression that can be used with FileMaker's Evaluate function.

Return Values

Returns the result of the expression.

Examples

```
// this example uses one of FileMaker's get functions
$expression = "Get ( ApplicationVersion )";

// print the expression and results
print("Example 1:\n" . $expression . " = " . fm_evaluate
($expression));
```

Returns:
Example 1:
Get (ApplicationVersion) = ProAdvanced 8.5v1

```
// this example returns the contents of a field
$expression = "GetField ( \"Test::Note\" )";

// print the expression and results
print("\n\nExample 2:\n" . $expression . " = " .
fm_evaluate($expression));
```

Returns:
Example 2:
GetField ("Test::Note") = Hi, I'm the contents of the note field.

```
// this example uses one of FileMaker's design functions
$expression = "Substitute ( ValueListItems ( Get
( FileName ) ; \"Colors\" ) ; \"¶\" ; \", \" )";

// print the expression and results
print("\n\nExample 3:\n" . $expression . " = " .
fm_evaluate($expression));
```

Returns:
Example 3:
Substitute (ValueListItems (Get (FileName) ; "Colors") ; "¶" ; ", ") = Red, Green, Blue

```
// this is an example of an invalid expression
// notice that the quotes are missing from the field
name
$expression = "GetField ( Test::Note )";

// print the expression and results
print($expression . " = " . fm_evaluate($expression));
```

Returns:
GetField (Test::Note) = ?

Script Functions

int **fm_perform_script** (string $file_name, string $script_name [, string $script_parameter [, int $control]])

Description

Used to perform FileMaker scripts.

Parameters

file_name (required)
The name of the file that contains the script to be performed.

script_name (required)
The name of the script to be performed.

script_parameter (optional)
Script parameter to be sent to the script.

control (optional)
Tells FileMaker what to do when other scripts are already running, 0 = halt, 1 = exit, 2 = resume, 3 = pause, which is the default.

Return Values

Returns one of the following FileMaker error codes.

0 No error
100 File is missing
104 Script is missing

Examples

```
// this example triggers a script named "Test Script"
// in the file named "FM_Perform_Script_Example"
// with a script parameter value of "Hello World!"
echo fm_perform_script("FM_Perform_Script_Example",
"Test Script", "Hello World!");
```

Returns:
0

bool **fm_script_control** ([bool $enable_disable])

Description

Allows script triggering to be disabled when needed. To disable, pass in false. To enable, pass in true. Using fm_script_control with no parameter will return the current setting, true if enabled, false if not enabled. The current setting is always returned, so enabling will return true, disabling will return false. By default, script triggering is enabled when the plug-in first starts up.

Parameters

enable_disable (optional)
The name of the file that contains the script to be performed.

Return Values

Returns **TRUE** or **FALSE** depending on the current setting.

Examples

```
// to disable script triggering
fm_script_control(false)
```

Returns:
FALSE

```
// to enable script triggering
fm_script_control(true)
```

Returns:
TRUE

```
// to return the current setting
fm_script_control()
```

Returns:
TRUE or FALSE depending on the current setting

Error Function

mixed **fm_get_last_error** ([int $selector])

Description

Used to check for errors returned by the “fm” functions included with the FileMaker PHP extension.

Parameters

selector (optional)

Use 1 to return only the error code or 2 to return only the error message. By default, both the error code and error message are returned.

Return Values

Returns the last error after calling one of the “fm” functions included with the PHP FileMaker extension. 0 is returned if there was no error. If only the error code is returned (using a selector value of 1) OR when there is no error, the return type will be int, otherwise the return type will be string.

Note that error numbers less than 1,000 are plug-in errors, while errors that are equal to or greater than 1,000 are user-defined errors (that have been set using the fm_set_error function).

Examples

```
echo fm_get_last_error();
```

Returns:

106 ERROR: Table missing (106) // return type is string

```
echo fm_get_last_error(1);
```

Returns:
106 (return type is int)

```
echo fm_get_last_error(2);
```

Returns:

ERROR: Table missing (106) (return type is string)

bool **fm_set_error**(int $error_number [, string $error_message])

Description

Used to set a user-defined error number and message. The error number and message can be retrieved using the plug-in's PHP_GetLastError function.

Note that error numbers below 1,000 are reserved for plug-in errors.

Parameters

error_number (required)
User defined error number. Must be equal to or greater than 1000.

error_message (optional)
A message explaining the error.

Return Values

Returns **FALSE** if an invalid *error_number* is specified, otherwise **TRUE**.

Examples

```
$error = fm_set_error(1000, 'The supplied URL is
invalid, please check the format and try again.');

var_dump($error);
```

Returns:
bool(true)

SQL Functions

mixed **fm_sql_select** (string $sql_query [, int $column_separator [, int $row_separator]])

Description

Used to perform an SQL select and return the selected data.

Calls the ExecuteSQL function in the FileMaker plug-in API. This function is considered experimental by FileMaker, so use with caution. See the FileMaker ODBC and JDBC Developer's Guide for reference to the supported SQL.

Parameters

sql_query (required)

Any supported SQL select query.

column_separator (optional)

ASCII or Unicode character code. The pipe character "|" (code 124) is the default.

row_separator (optional)

ASCII or Unicode character code. The return character "¶" (code 167) is the default.

Return Values

Returns the **results** of the select query or **FALSE** if an error occurs. Most errors are caused by invalid SQL statements. Use fm_get_last_error to get error details and keep in mind that the error that's returned will be a FileMaker error, so you can reference FileMaker help for more information.

Examples

```
echo fm_sql_select("SELECT * FROM Company WHERE
Status_ID = 'A'");
```

Returns:

All columns from the Company table for records that have a Status_ID of 'A'. Columns are separated by the default pipe character and rows are separated by the default return character.

bool **fm_sql_execute** (string $sql_query)

Description

Used to execute any supported SQL query **except CREATE, ALTER and DROP**. Use this function for INSERT, UPDATE and DELETE. Use fm_sql_select for SELECT.

Parameters

sql_query (required)
Any supported SQL query except SELECT, CREATE, ALTER or DROP.

It's very important that you do not attempt to execute an SQL query that includes CREATE, ALTER or DROP statements. This is not supported by this function and will require you to force quit FileMaker.

Return Values

Returns **TRUE** if execution is successful, otherwise **FALSE**.
Use fm_get_last_error to get error details and keep in mind that the error that's returned will be a FileMaker error, so you can reference FileMaker help for more information.

Examples

```
echo fm_sql_execute("UPDATE Company SET Name = 'Scodigo'
WHERE Company_ID = '1000'");
```

Returns:
TRUE

Variable Functions

mixed **fm_set_variable** (string $name, string $value)

Description

Used to set the value of a new or existing variable in the plug-in's memory.

These variables are different from standard FileMaker variables in that they are available "application-wide" (meaning that they can be accessed from any open FileMaker database).

Once a variable is created, it will persist in memory until 1) you delete it using fm_unset, 2) you quit FileMaker or 3) you stop the plug-in by un-checking it in the preferences.

All variables are stored as text so you may need to apply one of FileMaker's "GetAs" functions to convert variables back to the desired type.

Parameters

name (required)
The name of the variable to set. Note that any variable that begins with double underscores "__" will be considered hidden and will not be returned by the fm_get_defined_variables function.

value (required)
The value to set.

Return Values

Returns the **value** on success or **FALSE** on failure.

Examples

```
echo fm_set_variable('foo', '100');
```

Returns
100

mixed **fm_get_variable** (string $name)

Description

Used to get the value of an existing variable.

Parameters

name (required)
The name of the variable to get.

Return Values

Returns the **value** if the variable exists, otherwise **NULL**. Keep in mind that there's a difference between an empty string '' and NULL. It's perfectly legitimate for the value to be empty.

Examples

```
fm_get_variable('foo');
```

Returns:
100

```
fm_get_variable('bar');
```

Returns:
NULL

bool **fm_isset** (string $name)

Description

Used to determine whether a variable is set.

Parameters

name (required)
The name of the variable to check.

Return Values

Returns **TRUE** if the variable exists, otherwise **FALSE**.

Examples

```
echo fm_isset('foo');
```

Returns:
TRUE

```
echo fm_isset('bar');
```

Returns:
FALSE

bool **fm_unset** (string $name)

Description

Used to destroy the specified variable.

Parameters

name (required)
The name of the variable to destroy.

Return Values

Destroys the specified variable. Returns **TRUE** if the variable existed, otherwise **FALSE**.

Examples

```
echo fm_unset('foo');
```

Returns:
TRUE

```
echo fm_unset('bar');
```

Returns:
FALSE

array **fm_get_defined_variables** (void)

Description

Returns an array of all defined variables, along with their values. If no defined variables exist, or only hidden variables exist, an empty array is returned. Any variable that begins with double underscores "__" is considered hidden and will not be returned by this function. Use this naming convention for any variables that you don't want to be exposed by this function.

Parameters

None.

Return Values

Array.

Examples

```
print_r(fm_get_defined_variables());
```

Returns:

```
Array
(
   [foo] => 100
)
```

Thanks for using SmartPill!